Real-time Operating Systems and Systems Programming

Localization
mmap()

ARVUTITEADUSE INSTITUUT
DEPARTMENT OF COMPUTER SCIENCE

I18n ja L10n

- Internationalization – enabling translation support for a program
- Localization – translation and modifying a program to suit local idioms and customs

ASCII

- Time before ASCII luckily outside of our scope
- ASCII standard: characters with value of less than 32 are non-printable (bell sound or feeding a new paper into the printer)
- Characters above 127 free for anyone to use

IBM PC codepage (437)

- ASCII compatible
- For some European languages é and è letters
- Horizontal and vertical table-drawing characters
- Remember the older cashier screens
  - (For those you can use the curses library)
- What about Hebrew, Asian languages, Russian?

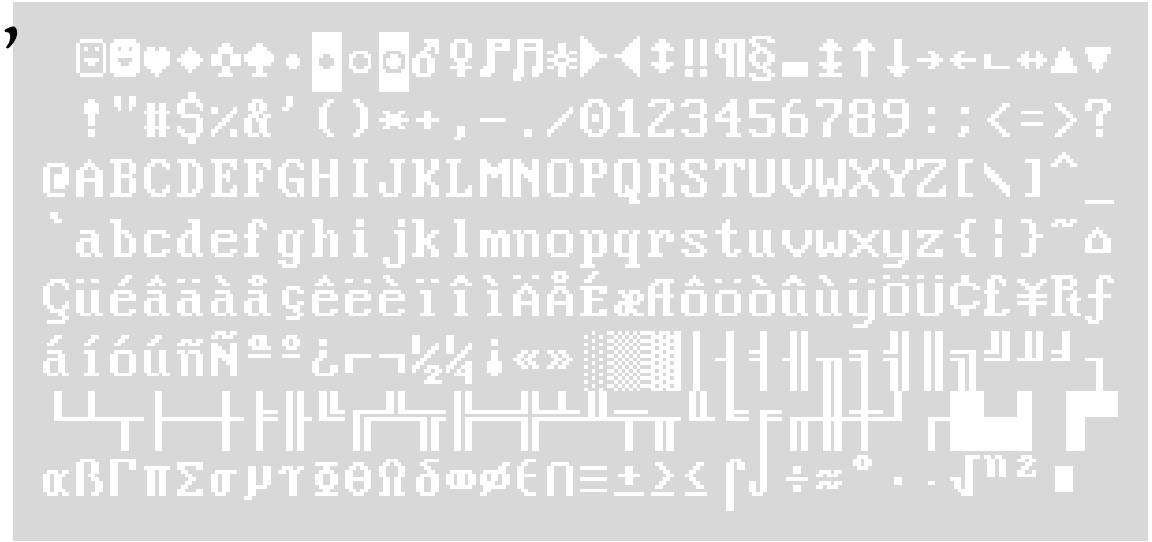


Illustration: https://en.wikipedia.org/wiki/File:Codepage-437.png

Code pages

- ASCII provides the base, upper characters different
- Support for several languages in parallel

- Asia uses two-byte codepage
- Result: the same computer can not display some languages in parallel (unless you create bitmap fonts for that specific purpose)
- IBMi ja Microsoft code-tables split after the end of their co-operation around 1990
- ANSI standard comes along too

Unicode to the rescue

- Unicode is a collection of *Code Points*
- Every *Code Point* refers to a symbol which sometimes is a character in some language like A or Õ, or something else like ffi (U+FB03)
- They exist in a rather plentiful manner (cat faces etc)
- You can refer to the *Code Points* using some specific encoding

Functionality

- Combining of letters
- ~ and o > õ combinations
- test
- Sign to swap text direction for right-to-left languages

Example

- test
- 74 65 73 74
- 2 byte: 00 74 00 65 00 73 00 74 (UCS-2 / UTF-16)
- Or? : 74 00 65 00 73 00 74 00
- FE FF : byte order mark
  - someone in Microsoft thought that it would be a good idea to put it before files and strings; Avoid in Unix world
- UCS-4 means 4-byte characters

Coding: UTF-8

- A specific coding
- Lower 127 characters are ASCII compatible
- Further bytes represent multibyte characters
- Linux has mostly completed standardization to UTF-8; Use of anything other than this should be considered problematic

Conclusions

- "Plain text" does not exist
- We are always interested in the encoding of the aforementioned "plain text"
  - Our Huffman encoder is also essentially a translation program from one encoding to another

In practice

- GNU library: libiconv
  http://www.gnu.org/software/libiconv/
- fopen("file.txt", "r, ccs=UTF-8");
- wchar_t data-type
- fgetc() >> wint_t fgetwc(FILE * stream)
- EOF >> WEOF

Linux support

- The input from the keyboard (what you get from terminal stdin) is converted to UTF-8 stream (console driver does this work)
- The output to console is decoded using a UTF-8 decoder and is presented using a 16-bit font
- BOM does not exist (the FE FF)

Two approaches

- Keep internal data in UTF-8
- Keep data in its decoded form and convert only upon outputting it
  - A character would be an object in memory in this case

Problems of internal UTF-8

- strlen() does not tell how many positions the cursor would move
- mbstowcs(NULL,s,0) returns the character count according to its coding

Usage

- Define locale in environment:
  LANG=et_EE (for output in ISO 8859-1)
  LANG=et_EE.UTF-8 (for output in UTF-8)
- #include <locale.h>
- setlocale() - LC_CTYPE or LC_ALL arguments
- command:
  locale -a shows the locales installed into system

Gettext

- Solutionf from Sun Microsystems
- Copied by GNU project
- Quite standard and widely used

Workflow

- Write your program using gettext() function and locale registration
- Use xgettext program to gather your strings into .pot file
- Create translation files for your target language using msginit command
- Translate
- Cinvert translation into binary using msgfmt program
- Put the result into /usr/share/locale/XX/LC_MESSAGES (XX is language; et or de, for example)

Hello.c

```
1     #include <libintl.h>
2     #include <locale.h>
3     #include <stdio.h>
4     #include <stdlib.h>
5     int main(void)
6     {
7       setlocale( LC_ALL, "" );
8       bindtextdomain( "hello", "/usr/share/locale" );
9       textdomain( "hello" );
10       printf( gettext( "Hello, world!\n" ) );
11       exit(0);
12     }
```

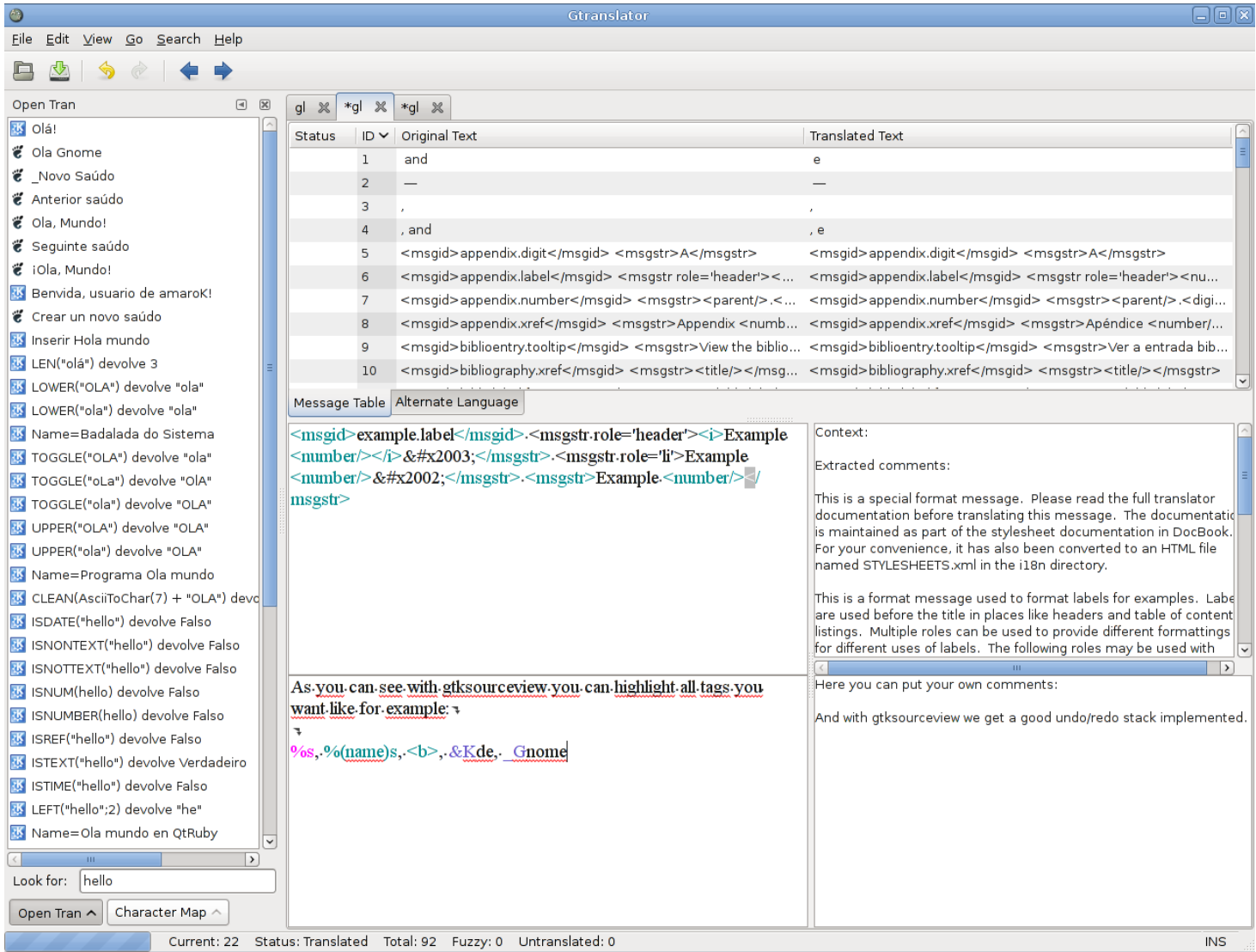
Source: http://oriya.sarovar.org/docs/gettext_single.html

Explanation

- setlocale() gathers the users preferences for language and its customs (date formats, week starting date, currency, etc)
- bindtextdomain() tells that „hello“ program can find its translation under /usr/share/locale (this is the default and could be skipped)
- textdomain() tells that language set is named "hello" in all of the languages
- **gettext()** should wrap all the strings; alias _

Tools

- gtranslator



Virtual memory

- Virtual memory gives you a large address space (from 0 to "really large", latter depending on system)
- It is not continous: every address is not usable
- Divided into pages. (usually 4kB)
- Every page is located in the memory (*frame*) or somewhere else (swap disk)
  - Emptied memory marked as: "contains zeroes"
- Virtual addresses are mapped to either real frames or swap space

Page fault

- As there is more virtual memory than real memory, we must swap pages between "real" and "backup" memory
- It's called paging
- Page fault: an attempt to read memory which is not in the RAM
- When page fault happens, the couple of milliseconds needed for memory access suddenly take a far greater amount of time
- Hard disks become noisy

How to get memory

- There are two ways of getting memory
  - upon starting your program (exec) when your program gets its memory space and is allocated space in there for its constants, code text and stack space
  - in your program:
    - auto variables
    - malloc
    - mmap: map a file into virtual memory
  - fork: *copy on write trikk*
- When program stops, its memory space collapses

Tracing memory

- You can trace memory allocation using

```
void mtrace(void);
void muntrace(void);
```

- Use environment variable named MALLOC_TRACE to specify the file which will store the statistics about memory allocation and release
- The first activates, the second deactivates trace
- GNU specific: mcheck.h provides it
- Result is not human-readable – use a command:

```
mtrace progamname mtrace-log
```

mmap()

- mmap() maps a file into virtual memory (or creates an anonymous mapping)
- Sometimes usefil:
  - We can read only parts of file which we use
  - mmap() lets you write changes back to disk
  - we can open files larger than mem+swap

```
void * mmap (void *address, size_t length, int protect,
             int flags, int filedes, off_t offset)
```

  - Parameters: desired start of mapping, length, protection data, management data, file descriptor and file offset

mmap() parameters

- prot: PROT_READ, PROT_WRITE, PROT_EXEC bits
  - depending on system: *write* is usually *read* or write protected files can not be written when PROT_READ is missing
- flags: refine mapping:
  - MAP_PRIVATE: don't write changes into file
  - MAP_SHARED: changes visible in file and other processes
  - MAP_FIXED: get this address or fail
  - MAP_ANONYMOUS: don't open a file (some systems expand heap using this trick)

mmap.c Example

munmap() & msync() & madvise()

- munmap(): removes mapped space starting from an addressto given address (may remove several); can handle unmapped segments.
- msync(): write mapping to file from given point
- madvise(): suggests how you want to use an address region: for random access, sequental access; will we need it all eventually or is the contents becoming irrelevant and when anything happens to it, the client won't leave the room in screaming agony.